华中科技大学

计算机科学与技术学院

《机器学习》结课报告

专　　业：数据科学与大数据技术

班　　级：大数据 2101 班

学　　号：U202115652

姓　　名：李嘉鹏

成　　绩：

指导教师：何琨

完成日期：2023 年 5 月 21 日

目录

1 机器学习第六次实验：鸢尾花三分类问题

1.1 实验要求

对于鸢尾花的三分类问题，需要对 Iris 数据集进行训练和测试，并根据提示补全 Python 代码。数据集包含 150 个样本，分为 3 类，分别是 Iris-setosa、Iris-versicolor 和 Iris-virginica，其中每类包含 50 个样本，而每个样本数据又包含 4 个属性特征。因此可以通过这 4 个属性特征预测鸢尾花属于三个种类中的哪一类。最终需要准确率达到 95%以上。

1.2 程序设计与实现

首先观察数据集特征，如图 1-1、图 1-2 所示，可以发现 150 个鸢尾花数据集被分为三组，每组包含 50 个数据，每个数据前四项为其特征。

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
```

图 1-1：Iris-setosa 数据集节选

```
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
5.7,2.8,4.5,1.3,Iris-versicolor
6.3,3.3,4.7,1.6,Iris-versicolor
```

图 1-2：Iris-versicolor 数据集节选

首先，要获取每个类的前五个样本信息。使用 load_iris 函数读取鸢尾花数据集，将其特征向量存储在 numpy 数组 x_data 中。接着，使用同一函数的 target 属性读取每个样本对应的标签（即类别），并将其变形为一个 150 行×1 列的数组 y_data。最后，使用 hstack 函数将 x_data 和 y_data 按列连接在一起，得到包含全部 150 个样本信息的二维数组 data。其中第 1-4 列分别对应鸢尾花四个特征，第 5 列对应类别标签。代码如下：

```
x_data = load_iris().data
y_data = load_iris().target.reshape(150, 1)
data = np.hstack((x_data, y_data))
```

接下来分别将 Iris-setosa、Iris-versicolor、Iris-virginica 对应为 0、1、2 三个标签。这里，可以定义一个字典 label_dict，用来存储类别与数字标签之间的映射关系，键值 0、1、2 分别对应三个类别。代码如下：

```
label_dict = {0: 0, 1: 1, 2: 2}
```

然后通过随机选择的方式将数据集分成训练集和验证集两部分，其中训练集占总数据集的 80%。先用 np.random.randint 函数生成长为 len(Y) * 0.8、包含了从 0 到 len(Y)-1 之间的整数的数组，也就是将数据集按照 8:2 的比例划分成训练集和验证集，该数组的元素代表了选出来的训练集样本在原数据集中所对应的下标索引。最后生成验证集的下标索引数组 test_idx。这一部分的代码如下：

```
train_idx = np.random.randint(0, len(Y), int(len(Y) * 0.8))
test_idx = np.array([i for i in range(0, len(Y)) if i not in train_idx])
```

下面将测试数据转换为 MindSpore 数据集。将 X_test 和 Y_test 打包成一个列表并赋值给变量 XY_test，并使用 GeneratorDataset 函数创建一个 MindSpore 数据集 ds_test，该数据集的两个字段分别对应 X_test 和 Y_test。最后仿照原始程序，使用 shuffle 函数打乱 ds_test，利用大小为 120 的缓冲区进行存储，并使用 batch 函数对 ds_test 进行分批处理，每批大小为 32。代码如下：

```
XY_test = list(zip(X_test, Y_test))
ds_test = dataset.GeneratorDataset(XY_test, ['x', 'y'])
ds_test = ds_test.shuffle(buffer_size=120).batch(32, drop_remainder=True)
```

接下来，使用交叉熵损失计算。定义损失函数 SoftmaxCrossEntropyWithLogits，采用均值作为损失函数的缩放因子来计算平均损失。同时使用动量优化器优化参数，learning_rate=0.05 表示学习率设置为 0.05，即每次更新参数时的步长大小；momentum=0.9 表示动量设置为 0.9，即在参数更新时保持历史梯度的加权平均。代码如下：

```
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
opt = nn.optim.Momentum(net.trainable_params(), learning_rate=0.05,
momentum=0.9)
```

1.3 实验环境与平台

本次实验在华为云实验平台完成。在华为云 ModelArts 平台上创建 AI 框架为 Mindspore-1.5、硬件环境为 Ascend 910+ARM 的开发环境，并创建好搭建了 MindSpore 环境的 Jupyter Notebook，在线上填入代码运行并测试。

1.4 结果与分析

运行程序，使用 model.train 进行模型的训练，迭代 25 次，每个迭代周期结束时都对测试集进行验证并输出测试集上的损失函数值，如图 1-3 所示。最终的准确率为 0.98438，损失函数值为 0.1665，达到了预期目标，预测效果较好。

```
epoch: 1 step: 3, loss is 1.0233852863311768
epoch: 2 step: 3, loss is 0.8723608255386353
epoch: 3 step: 3, loss is 0.6527835130691528
epoch: 4 step: 3, loss is 0.4745059907436371
epoch: 5 step: 3, loss is 0.451712429523468
epoch: 6 step: 3, loss is 0.4051048755645752
epoch: 7 step: 3, loss is 0.366272509098053
epoch: 8 step: 3, loss is 0.46162301301956177
epoch: 9 step: 3, loss is 0.29690998792648315
epoch: 10 step: 3, loss is 0.3816644549369812
epoch: 11 step: 3, loss is 0.2652394771575928
epoch: 12 step: 3, loss is 0.26634204387664795
epoch: 13 step: 3, loss is 0.3055557906627655
epoch: 14 step: 3, loss is 0.2017630636692047
epoch: 15 step: 3, loss is 0.20544181764125824
epoch: 16 step: 3, loss is 0.18622274696826935
epoch: 17 step: 3, loss is 0.1978479027748108
epoch: 18 step: 3, loss is 0.19512839615345
epoch: 19 step: 3, loss is 0.1940665990114212
epoch: 20 step: 3, loss is 0.21145717799663544
epoch: 21 step: 3, loss is 0.22932618856430054
epoch: 22 step: 3, loss is 0.16090518236160278
epoch: 23 step: 3, loss is 0.2618800401687622
epoch: 24 step: 3, loss is 0.19985099136829376
epoch: 25 step: 3, loss is 0.21769745647907257
{'acc': 0.984375, 'loss': 0.16646874696016312}
```

图 1-3：鸢尾花三分类问题测试结果

1.5 个人体会

这次实验相对来说比较容易实现，主要是要理解题目要我达到什么目的。鸢尾花数据集是机器学习领域公认的入门数据集，在进行鸢尾花三分类问题的实验中，我首先了解了 Iris 数据集的特征和标签分布情况，并使用 Python 中的 sklearn 库进行数据处理和模型训练。通过在 MindSpore 下使用交叉熵损失计算，得到了一个准确率较高的分类器。

这次实验让我更深入地理解了机器学习的基本原理和实际应用过程，也提高了我对 Python 编程和数据分析的技能水平。

2 机器学习第六次实验：MNIST 手写数据集识别

2.1 实验要求

对于 MNIST 手写数字识别，需要对 60000 个训练集数据（手写数字的图片）进行学习，并在 10000 个测试集数据上测试，准确率要达到 95%以上。

2.2 程序设计与实现

首先对原始数据集进行放缩和归一化，将图像的高度和宽度都设为 256，并使用除以 255 的方式对图像进行归一化操作，将其值转换到 0~1 之间。代码是：

```
resize_height = 32
resize_width = 32
rescale = 1/255.
```

接下来对数据集进行打乱（batch）、分批（shuffle）和重复（repeat）操作，以便后续训练模型时使用。调用 shuffle 函数对数据集进行打乱，接着调用 batch 函数将每一批的数据打包成一个 batch，其中 drop_remainder=True 表示丢弃最后不足一个 batch 的样本。最后调用 repeat 函数对整个数据集重复操作，重复 repeat_size 次，这样就可以在训练模型时多次使用该数据集进行迭代更新。代码如下：

```
mnist_ds = mnist_ds.shuffle(buffer_size=buffer_size)
mnist_ds = mnist_ds.batch(batch_size, drop_remainder=True)
mnist_ds = mnist_ds.repeat(repeat_size)
```

建立 LeNet5 网络结构神经网络，搭建神经网络时定义一些层和模块。首先定义两个卷积层，其中 self.conv1 的输出通道数为 6；self.conv2 的输入通道数为 6，输出通道数为 16，二者的卷积核大小均为 5×5。接着定义三个全连接层，其中 self.fc1 的输入节点数为 400，输出节点数为 120，采用正态分布初始化；self.fc2 的输入节点数为 120，输出节点数为 84；self.fc3 的输入节点数为 84，输出节点数为 num_class，即分类数。

最后，定义一个 ReLU 层 self.relu、一个步长为 2 的最大池化层 self.max_pool2d、一个将输入展平为一维向量的层的池化窗口 self.flatten。代码如下：

```
self.conv1 = nn.Conv2d(num_channel, 6, 5, pad_mode='valid')
self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
self.fc1 = nn.Dense(16 * 5 * 5, 120, weight_init=Normal(0.02))
self.fc2 = nn.Dense(120, 84, weight_init=Normal(0.02))
```

```
self.fc3 = nn.Dense(84, num_class, weight_init=Normal(0.02))
self.relu = nn.ReLU()
self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
self.flatten = nn.Flatten()
```

下面要使用上一步定义好的运算构建前向网络。首先将输入 x 经过第一个卷积层 conv1，使用激活函数进行激活，再经过最大池化操作得到新的特征图 x。将这个 x 再经过第二个卷积层 conv2，重复上述操作，得到另一个新的特征图 x。对其进行扁平化操作并转化为向量，方便之后的全连接层进行处理。

然后，将上一步得到的向量 x 作为输入，依次经过三个全连接层进行线性变换，得到最终的输出结果。这样就实现了神经网络模型的前向传播过程，将输入数据 x 从输入层经过一系列的卷积、池化和全连接后输出。核心代码是：

```
x = self.max_pool2d(self.relu(self.conv1(x)))
x = self.max_pool2d(self.relu(self.conv2(x)))
x = self.flatten(x)
x = self.relu(self.fc1(x))
x = self.relu(self.fc2(x))
x = self.fc3(x)
```

最后只需分别定义训练和验证的方法。训练过程中，每次迭代结束后都输出损失函数的值 loss 和梯度更新次数 step，最后验证环境输出准确率 acc。

2.3 实验环境与平台

本次实验在华为云实验平台完成。在华为云 ModelArts 平台上创建 AI 框架为 Mindspore-1.5、硬件环境为 Ascend 910+ARM 的开发环境，并创建好搭建了 MindSpore 环境的 Jupyter Notebook，在线上填入代码运行并测试。

2.4 结果与分析

运行程序，总共执行 1875 次梯度更新，每次迭代结束时都对测试集进行验证并输出测试集上的损失函数值，如图 2-1 所示。最终的准确率为 0.96915，损失函数值为 0.0841，达到了预期目标，训练效果较好。

```
epoch: 1 step: 125, loss is 2.3063466548919678
epoch: 1 step: 250, loss is 2.3013103008270264
epoch: 1 step: 375, loss is 2.3123466968536377
epoch: 1 step: 500, loss is 2.3128714561462402
epoch: 1 step: 625, loss is 2.2944419384002686
epoch: 1 step: 750, loss is 2.2910056114196777
epoch: 1 step: 875, loss is 2.3012802600860596
epoch: 1 step: 1000, loss is 2.280142307281494
epoch: 1 step: 1125, loss is 0.6782105565071106
epoch: 1 step: 1250, loss is 0.26893293857574463
epoch: 1 step: 1375, loss is 0.6209826469421387
epoch: 1 step: 1500, loss is 0.37599292397499084
epoch: 1 step: 1625, loss is 0.20636290311813354
epoch: 1 step: 1750, loss is 0.033131662756204605
epoch: 1 step: 1875, loss is 0.08406174182891846
{'Accuracy': 0.9691506410256411}
Predicted: "4", Actual: "4"
```

图 2-1：MNIST 手写数字识别测试结果

2.5 个人体会

在这次实验中，我对 60000 个训练集数据进行了学习，并在 10000 个测试集数据上进行了测试。我建立了 LeNet5 神经网络，并通过 1875 次梯度更新来优化模型。

通过本次实验，我深刻认识到了数据处理的重要性。对于模型训练而言，数据的质量和数量是决定模型效果的关键因素之一。同时，我也学会了如何利用深度学习框架 TensorFlow 来建立神经网络模型，并掌握了梯度下降算法的基本原理和使用方法。总之，我不仅学到了理论知识，还锻炼了实践操作能力，对未来的学习有很大帮助。

3 机器学习大作业：Kaggle 2017 年房价预测（House Prices - Advanced Regression Techniques）

3.1 实验要求

大作业是 Kaggle 平台上的一个竞赛题目，主要是根据给定的房价数据集（包含影响房价的各种因素以及最终的房价），预测具有同样属性集的测试集的房价（链接：https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/code?competitionId=5407）。

具体来说，题目给定了 79 个房价的影响因素，要求对测试集中的每个 id 预测其房价。最终将预测结果提交到 Kaggle 上评测，评分采用均方误差 RMSE 表示，这个值越低越好（代表预测结果越准确）。

3.2 设计与实现

大作业的整体流程是：对数据进行清洗和预处理；将类别属性数值化并利用随机森林（Ramdom Forest）实现特征选择；进行数据的相关性分析、方差分析与标准化；通过主成分分析 PCA 进行降维；利用 Kmeans 算法实现聚类（二分类）；训练、线性回归并得到预测结果。

3.2.1 数据集特点分析

首先分析给定的数据集。数据集以 csv 的格式呈现，文件名为“test.csv”，其中包含了全部 79 个影响房价的属性以及每个 id 对应的房价，如图 3-1 所示。

	A	B	C	D	E	F	G	H	I	J	K
1	Id	MSSubCla	MSZoning	LotFrontag	LotArea	Street	Alley	LotShape	LandContc	Utilities	LotConfig
2	1	60	RL	65	8450	Pave	NA	Reg	Lvl	AllPub	Inside
3	2	20	RL	80	9600	Pave	NA	Reg	Lvl	AllPub	FR2
4	3	60	RL	68	11250	Pave	NA	IR1	Lvl	AllPub	Inside
5	4	70	RL	60	9550	Pave	NA	IR1	Lvl	AllPub	Corner
6	5	60	RL	84	14260	Pave	NA	IR1	Lvl	AllPub	FR2
7	6	50	RL	85	14115	Pave	NA	IR1	Lvl	AllPub	Inside
8	7	20	RL	75	10084	Pave	NA	Reg	Lvl	AllPub	Inside
9	8	60	RL	NA	10382	Pave	NA	IR1	Lvl	AllPub	Corner
10	9	50	RM	51	6120	Pave	NA	Reg	Lvl	AllPub	Inside
11	10	190	RL	50	7420	Pave	NA	Reg	Lvl	AllPub	Corner
12	11	20	RL	70	11200	Pave	NA	Reg	Lvl	AllPub	Inside
13	12	60	RL	85	11924	Pave	NA	IR1	Lvl	AllPub	Inside
14	13	20	RL	NA	12968	Pave	NA	IR2	Lvl	AllPub	Inside
15	14	20	RL	91	10652	Pave	NA	IR1	Lvl	AllPub	Inside
16	15	20	RL	NA	10920	Pave	NA	IR1	Lvl	AllPub	Corner

图 3-1：房价数据集预览

数据集共 1461 行，除第一行为属性名称外，每一行表示一个房屋售价样本，包含 79 个属性和 1 个房价值。其中这 79 个属性的说明如表 3-1 所示（其中数值属性的类别标记为 1，类别属性标记为 2）。

表 3-1：影响房价的 79 个属性说明

#	属性名	类别	含义	#	属性名	类别	含义
1	MSSubClass	1	建筑类别	**41**	TotalBsmtSF	1	地下室总面积
2	MSZoning	2	分区分类	**42**	Electrical	2	电气系统
3	LotFrontage	1	街道长度	**43**	1stFlrSF	1	一楼面积
4	LotArea	1	地块面积	**44**	2ndFlrSF	1	二楼面积
5	Street	2	道路通行方式	**45**	LowQualFinSF	1	低质量完成面积
6	Alley	2	巷道通行方式	**46**	GrLivArea	1	地上起居面积
7	LotShape	2	财产类型	**47**	BsmtFullBath	1	地下室全浴室
8	LandContour	2	财产平整度	**48**	BsmtHalfBath	1	地下室半浴室
9	Utilities	2	公共设施	**49**	FullBath	1	地上全浴室
10	LotConfig	2	地块配置	**50**	HalfBath	1	地上半浴室
11	LandSlope	2	倾斜程度	**51**	BedroomAbvGr	1	卧室数量
12	Neighborhood	2	地理位置	**52**	KitchenAbvGr	1	厨房数量
13	Condition1	2	靠近铁路	**53**	KitchenQual	2	厨房质量
14	Condition2	2	靠近铁路	**54**	TotRmsAbvGrd	1	总房间数
15	BldgType	2	住宅类型	**55**	Functional	2	功能评级
16	HouseStyle	2	住宅风格	**56**	Fireplaces	1	壁炉数量
17	OverallQual	1	总体装修质量	**57**	FireplaceQu	2	壁炉质量
18	OverallCond	1	总体评级	**58**	GarageType	2	车库位置
19	YearBuilt	1	建造年份	**59**	GarageYrBlt	1	车库建造年份
20	YearRemodAdd	1	改造年份	**60**	GarageFinish	2	车库装修
21	RoofStyle	2	屋顶类型	**61**	GarageCars	1	车位大小

22	RoofMatl	2	屋顶材料	**62**	GarageArea	1	车库面积
23	Exterior1st	2	外部材料	**63**	GarageQual	2	车库质量
24	Exterior2nd	2	外部材料	**64**	GarageCond	2	车库条件
25	MasVnrType	2	饰面类型	**65**	PavedDrive	2	铺设车道
26	MasVnrArea	1	饰面面积	**66**	WoodDeckSF	1	木板面积
27	ExterQual	2	外材质量	**67**	OpenPorchSF	1	露台面积
28	ExterCond	2	外材条件	**68**	EnclosedPorch	1	封闭露台面积
29	Foundation	2	地基类型	**69**	3SsnPorch	1	三季阳光房面积
30	BsmtQual	2	地下室高	**70**	ScreenPorch	1	带纱窗的阳光房面积
31	BsmtCond	2	地下状况	**71**	PoolArea	1	泳池面积
32	BsmtExposure	2	地下室墙体类别	**72**	PoolQC	2	泳池质量
33	BsmtFinType1	2	地下室装修质量	**73**	Fence	2	围栏质量
34	BsmtFinSF1	1	完成面积	**74**	MiscFeature	2	杂项功能
35	BsmtFinType2	2	地下室装修质量	**75**	MiscVal	1	杂项价值
36	BsmtFinSF2	1	完成面积	**76**	MoSold	1	出售月份
37	BsmtUnfSF	1	未完成面积	**77**	YrSold	1	出售年份
38	Heating	2	供暖方式	**78**	SaleType	2	销售类型
39	HeatingQC	2	供暖质量	**79**	SaleCondition	2	销售条件
40	CentralAir	2	中央空调	**-**	SalePrice	1	最终售价

观察数据集可知以下三点：

（1） 数据集中影响房价的因素太多，需要找到影响房价的主要属性。

（2） 属性可以被分为两部分：数值属性（用数字进行量化，共 37 种）和类别属性（用类别表示，共 42 种）。对类别属性，考虑使用某种方法将其转变为数值属性，方便处理。例如，上表中属性 CentralAir 只存在两种取值 Y 或 N，据此可以将其替换为 1 或 0。

（3） 很多属性的列中存在大量空值 NAN，不利于后续训练，需要进行数据的清洗和预处理。

3.2.2 数据清洗与预处理

首先将所有形如""、"--"、"?"、"na"、"NAN"、"nan","的值都视为空值（NAN），并将其替换为空。这样就完成了对空值的集中处理。

此时还没有将类别属性数值化，故先分析数值属性。绘制训练集全部数值属性相关性示意图，如图 3-2 所示。

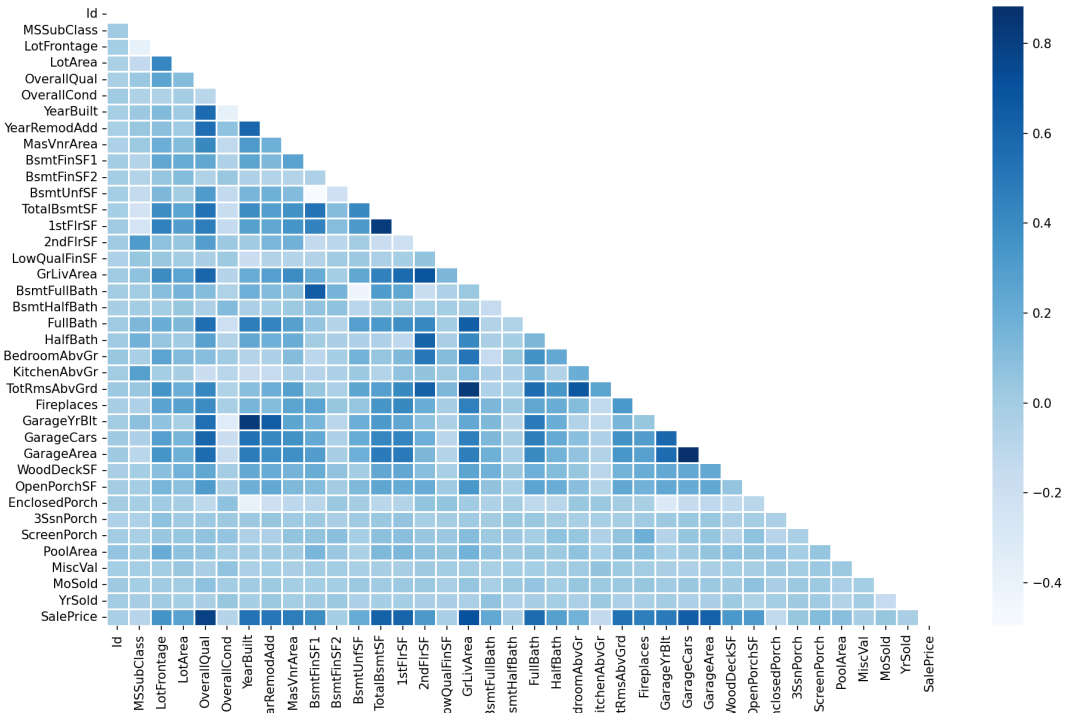

图 3-2：训练集数值属性相关性示意图

上图中，颜色越深代表两种属性间的相关性越强。可见不同的属性对对房价的影响程度不同。

接下来，再进一步绘制每个属性的空值密集程度图，如图 3-3 所示。若某个属性包含很多空值，则表现为大量空白，反之则不会出现大量空白。

可见，大部分属性的空值出现次数较少，而少部分属性则存在极大空缺，这部分属性需要被移除掉，以避免出现数据偏差和模型性能下降。在实际操作中，我统计了每个属性的空缺值，若超过 600 个值空缺则将其视为无关变量，在后续学习中不再考虑这些因素的影响。这些属性是：Alley、FireplaceQu、PoolQC、Fence、MiscFeature。

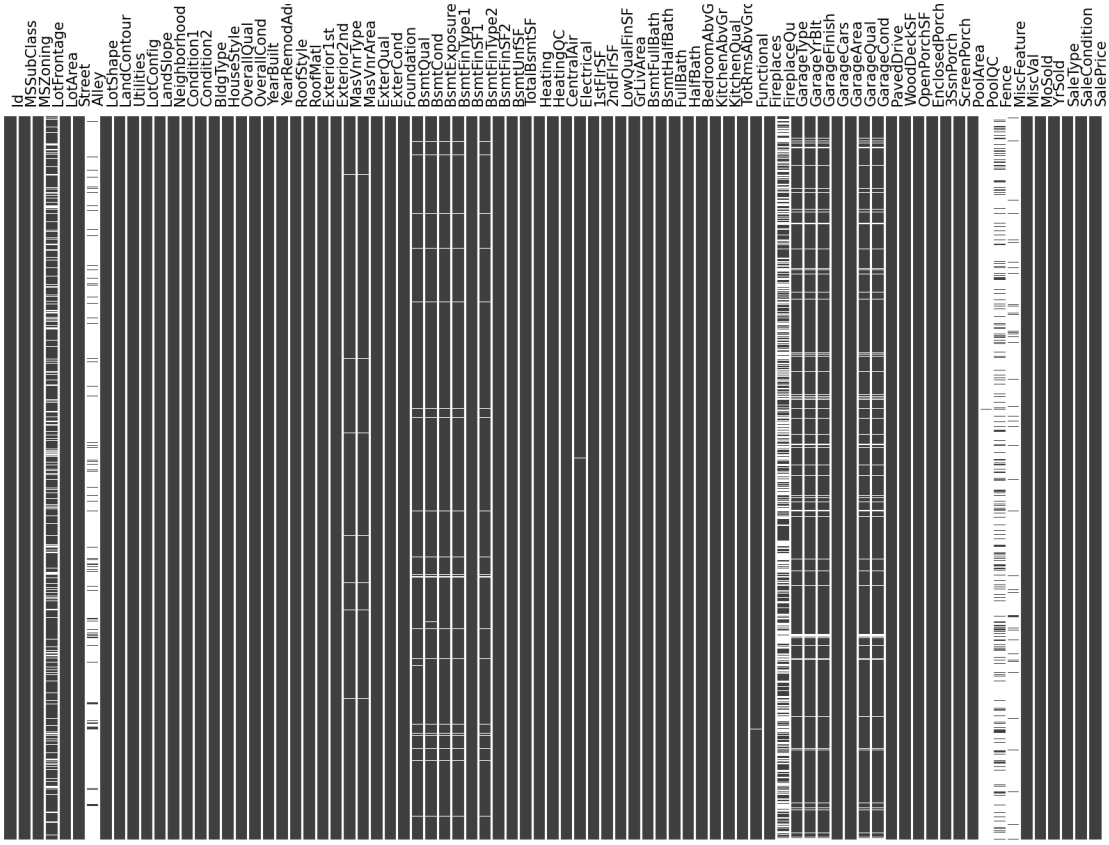

图 3-3：训练集空值密集程度图

下面对剩余列的缺失值进行填充，其中数值属性用这一列的平均值填充，类别属性用这一列出现次数最多的属性填充，代码如下。

```
for attributes in full_dataset:
    if attributes in numerical_attr:
        full_dataset[attributes] =
full_dataset[attributes].fillna(value=full_dataset[attributes].median())
    else:
        full_dataset[attributes] =
full_dataset[attributes].fillna(value=full_dataset[attributes].value_count
s().idxmax())
```

3.2.3 将类别属性数值化并利用随机森林实现特征选择

上面已经得到了类别属性的总数为 42。为了从这么多的属性中获得相对重要的类别属性及其对应的取值，这里需要引入 sklearn 库中的随机森林分类器 RandomForestClassifier、特征选择器 SelectFromModel。同时为了避免过拟合，需要引入 train_test_spilt 函数对训练集和测试集进行分割。

首先，从 train_data 中取得类别属性，使用 pd.get_dummies 对每个类别属性进行独热编码，转化成二进制数值。这样可以将类别属性转化为编码。通过循环遍历每个属性，将所有的类别属性都转换成了数值属性。接着使用 train_test_split 函数将数据集分割成训练集和测试集，比例为 3:1。核心代码如下：

```
for at in X:
    X = pd.concat([X, pd.get_dummies(X[at], prefix=at, drop_first=True)],
axis=1, sort=False)
    X.drop(columns=[at], inplace=True)

X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X, Y,
test_size=0.25)
```

接下来通过随机森林分类器和特征选择器对类别属性进行筛选，保留最重要的类别属性。创建 150 棵树进行随机森林处理，并利用 RFC 作为估计器，选取重要度大于全部类别属性取值重要度平均值的类别属性取值。

```
rfc = RandomForestClassifier(n_estimators=150)
sel = SelectFromModel(estimator=rfc, threshold='mean')
sel.fit(X, Y)

print("门限 threshold: ", sel.estimator_.feature_importances_.mean())
print(sel.get_support())

selected_features = X_train_st.columns[(sel.get_support())]
print("高于门限值的类别属性取值个数：", len(selected_features), "\n 这些属性的值
是：")
print(selected_features)
```

程序输出了门限值（即重要度的平均值）以及高于门限值的全部类别属性取值、全部类别属性取值的重要度，如图 3-4、图 3-5 所示。可见，有 73 个类别属性的具体取值（注意不是 73 个属性）高于门限值。

此后，为了进一步找出重要的类别属性，还要将高于门限值的类别属性取值的重要度从高到低排序并输出，同时绘制重要度排名前十的类别属性取值直方图，如图 3-6、图 3-7 所示。

```
门限threshold:  0.00510204081632653
高于门限值的类别属性取值个数： 72
这些属性的值是：
Index(['MSZoning_RL', 'MSZoning_RM', 'LotShape_Reg', 'LandContour_Lvl',
       'LotConfig_CulDSac', 'LotConfig_FR2', 'LotConfig_Inside',
       'Neighborhood_CollgCr', 'Neighborhood_Edwards', 'Neighborhood_Gilbert',
       'Neighborhood_NAmes', 'Neighborhood_NridgHt', 'Neighborhood_OldTown',
       'Neighborhood_Sawyer', 'Neighborhood_SawyerW', 'Condition1_Feedr',
       'Condition1_Norm', 'BldgType_TwnhsE', 'HouseStyle_1Story',
       'HouseStyle_2Story', 'HouseStyle_SLvl', 'RoofStyle_Gable',
       'RoofStyle_Hip', 'Exterior1st_HdBoard', 'Exterior1st_MetalSd',
       'Exterior1st_Plywood', 'Exterior1st_VinylSd', 'Exterior1st_Wd Sdng',
       'Exterior2nd_HdBoard', 'Exterior2nd_MetalSd', 'Exterior2nd_Plywood',
       'Exterior2nd_VinylSd', 'Exterior2nd_Wd Sdng', 'MasVnrType_BrkFace',
       'MasVnrType_None', 'MasVnrType_Stone', 'ExterQual_Gd', 'ExterQual_TA',
       'ExterCond_Gd', 'ExterCond_TA', 'Foundation_CBlock', 'Foundation_PConc',
       'BsmtQual_Gd', 'BsmtQual_TA', 'BsmtCond_Gd', 'BsmtCond_TA',
       'BsmtExposure_Gd', 'BsmtExposure_Mn', 'BsmtExposure_No',
       'BsmtFinType1_BLQ', 'BsmtFinType1_GLQ', 'BsmtFinType1_LwQ',
       'BsmtFinType1_Rec', 'BsmtFinType1_Unf', 'BsmtFinType2_Unf',
       'HeatingQC_Gd', 'HeatingQC_TA', 'Electrical_SBrkr', 'KitchenQual_Gd',
       'KitchenQual_TA', 'Functional_Typ', 'GarageType_Attchd',
       'GarageType_BuiltIn', 'GarageType_Detchd', 'GarageFinish_RFn',
       'GarageFinish_Unf', 'GarageQual_TA', 'GarageCond_TA', 'PavedDrive_Y',
       'SaleType_WD', 'SaleCondition_Normal', 'SaleCondition_Partial'],
      dtype='object')
```

图 3-4：高于门限值的全部类别属性取值

```
全部类别属性取值的重要度： [3.05539741e-03 1.62611814e-03 9.36756565e-03 7.01134579e-03
 6.65227259e-04 5.08402539e-03 1.55630820e-03 2.64595269e-02
 4.12058307e-03 2.74688892e-03 8.12686931e-03 1.80649118e-04
 9.33146533e-03 5.95596892e-03 1.07537821e-03 2.44418825e-02
 4.58886986e-03 9.57027979e-04 2.86802515e-04 1.00508327e-03
 4.28929152e-03 2.44651648e-03 1.03127911e-02 4.10751037e-03
 6.77579889e-03 6.04461855e-03 2.89698364e-03 1.06939767e-03
 4.74053790e-03 1.02349879e-02 9.53122126e-04 4.98278026e-03
 3.72543463e-03 5.70477175e-03 5.40676026e-03 2.32159112e-03
 5.82984415e-03 5.49318486e-03 4.43725075e-03 2.40077851e-03
 4.48265585e-03 1.41737010e-03 6.02356076e-03 1.16074459e-02
 9.28963906e-04 2.29051662e-03 1.41713913e-03 3.21033366e-03
 3.82755425e-04 6.33734461e-04 6.47872949e-04 1.17294119e-03
 1.42754652e-04 3.84345910e-04 1.17106582e-04 2.60217769e-04
 2.16455147e-04 2.39087158e-03 3.31610388e-03 4.29603060e-03
```

图 3-5：全部类别属性取值的重要度（节选）

这些类别属性取值的重要度：

	Features	Importance
7	LotShape_Reg	0.026460
15	LotConfig_Inside	0.024442
171	GarageFinish_RFn	0.024307
132	BsmtExposure_No	0.023256
137	BsmtFinType1_Unf	0.021692
..	...	...
52	Condition2_PosA	0.000143
54	Condition2_RRAe	0.000117
164	Functional_Sev	0.000113
128	BsmtCond_Po	0.000086
155	Electrical_Mix	0.000057

图 3-6：类别属性取值的重要度（从高到低排序）

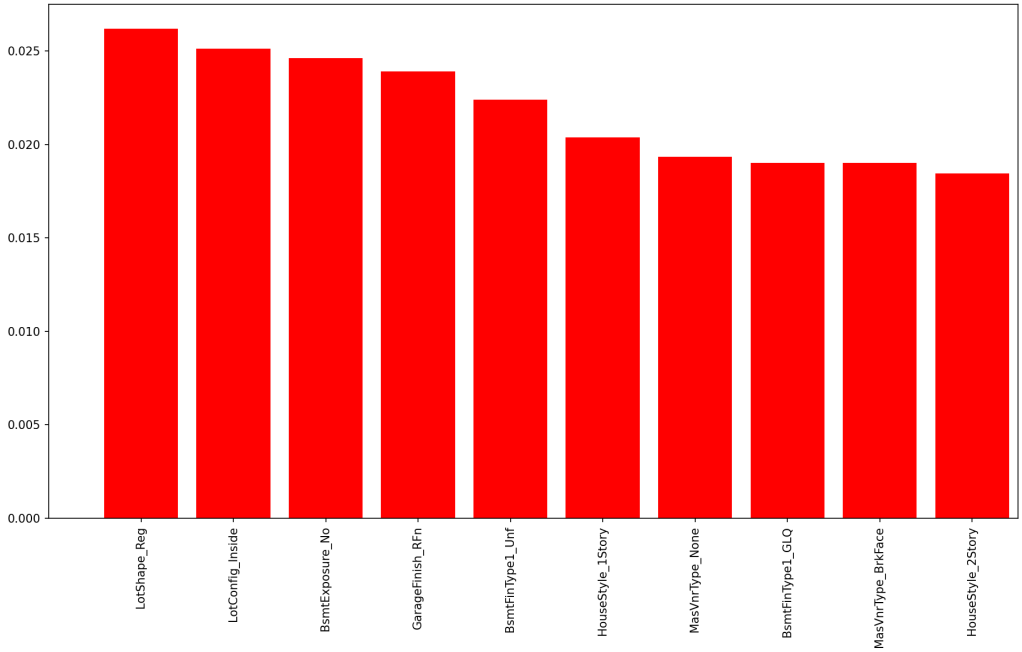

图 3-7：重要度排名前十的类别属性取值直方图

对这些值所属的类别属性进行数值化，分别是：GarageFinish、LotShape、BsmtExposure、BsmtFinType1、LotConfig、HouseStyle、MasVnrType。其中以 GarageFinish 属性为例，数值化后的效果如图 3-8 所示。即在该类别属性上，两种取值 RFn 和 Unf 分别变为新的两列，并做 0-1 化处理。

类别属性： GarageFinish

	Id	MSSubClass	MSZoning	...	SalePrice	GarageFinish_RFn	GarageFinish_Unf
0	1	60	RL	...	208500.0	1	0
1	2	20	RL	...	181500.0	1	0
2	3	60	RL	...	223500.0	1	0
3	4	70	RL	...	140000.0	0	1

图 3-8：类别属性数值化（以 GarageFinish 属性为例）

至此，已经完成了重要的特征选择工作，并可以画出完整的训练集属性相关性示意图，如图 3-9 所示。其含义已在前一部分解释过，此处不再赘述。

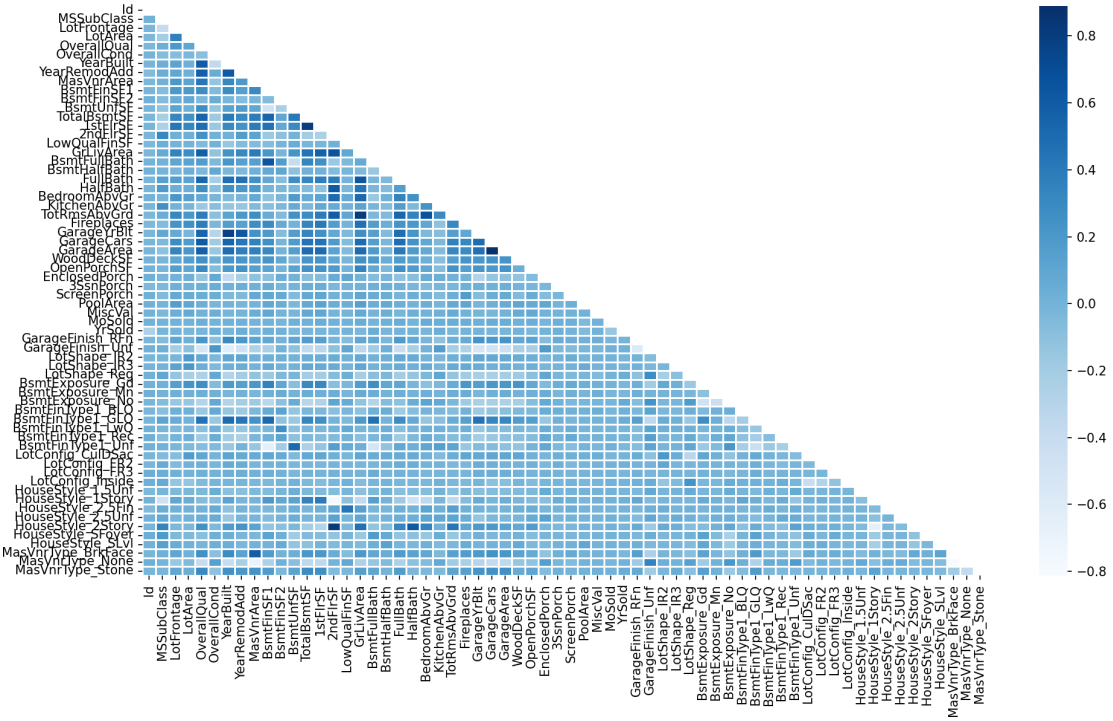

图 3-9：完整的训练集属性相关性示意图

3.2.4 数据的相关性分析、方差分析与标准化

经过上一步操作，已经得到了需要处理的全部属性的数值形式。

但关键问题在于：各个属性之间的值域差别太大，有的取值范围是在 0 和 1 之间二选一，有的是在 1~10 之间，还有的是在 0~2000 之间。巨大的尺度差异会在后续聚类过程中放大某些因素对聚类的影响，同时会在很大程度上忽略某些因素。为此，需要对数据进行标准化，使各个维度的数据在决定各个数据点的特征上具有同等地位。因此下面首先对数据进行相关性分析和方差分析。

（1）相关性分析

计算每种属性与房价 Saleprice 的相关系数，并创建一个名为 CORR 的列保存，如图 3-10 所示。可以看到一些属性的相关系数为正，代表其与房价成正相关；反之，若相关系数为负，代表其与房价成负相关。相关系数的绝对值越大，代表其对房价的影响越大。

属性相关性分析：

GrLivArea	10.331477
OverallQual	8.758551
TotRmsAbvGrd	8.232954
GarageArea	8.150476
GarageCars	8.119840
	...
HouseStyle_1Story	-1.966004
LotShape_Reg	-2.960485
BsmtExposure_No	-3.645429
MasVnrType_None	-5.080620
GarageFinish_Unf	-5.776358

图 3-10：全部属性对房价的相关系数

（2）方差分析和标准化

遍历每一个属性，计算其平均值、方差与标准化方差（标准差）并输出结果，如图 3-11 所示。然后导入 scipy.stats 库中的 zscore 函数来对数据进行标准化。使用 zscore 函数进行标准化后得到的是一个标准正态分布，即均值为 0、标准差为 1 的分布，有利于后续聚类操作。Zscore 的计算公式如下。

$$zscore = \frac{x - \mu}{\sigma}$$

其中，x是原始数据，μ是原始数据的平均值，σ是原始数据的标准差。

方差分析：

	属性	平均值	方差	标准化方差
3	LotArea	10168.114080	6.220471e+07	7886.996359
0	Id	1460.000000	7.102900e+05	842.787043
34	MiscVal	50.825968	3.219453e+05	567.402211
16	GrLivArea	1500.759849	2.560877e+05	506.051045
9	BsmtFinSF1	441.398253	2.075119e+05	455.534750
..	...	...	...	...
57	HouseStyle_2.5Unf	0.008222	8.157187e-03	0.090317
54	HouseStyle_1.5Unf	0.006509	6.468926e-03	0.080430
40	LotShape_IR3	0.005481	5.453152e-03	0.073845
52	LotConfig_FR3	0.004796	4.774796e-03	0.069100
56	HouseStyle_2.5Fin	0.002741	2.734090e-03	0.052289

图 3-11：方差分析

实现数据标准化的代码是：

```
from scipy.stats import zscore
old_full_dataset = full_dataset
full_dataset[numerical_attr] = zscore(full_dataset[numerical_attr])
```

3.2.5 通过主成分分析 PCA 进行降维

上面得到的属性种数仍然过多，因此考虑使用主成分分析（PCA）对数据进行降维，其原理解释如下。

在 PCA 中，可解释方差贡献率是指每个主成分能够解释原始数据的方差占比。它表示每个主成分对于原始数据的重要程度，即在保持数据尽可能多的信息的同时，减少数据的维数。通常情况下会优先选择保留解释方差贡献率较高的主成分，以尽可能地保留原始数据的信息。

根据这一原理，首先导入 sklearn.decomposition 中的 PCA 模块并 fit 该模型，然后计算各个属性的可解释方差贡献率（表示每个属性解释数据的方差占比）并输出，如图 3-12 所示。

```
PCA的各特征（可解释性）方差贡献率：
[1.41320325e-01 7.56189932e-02 4.75210457e-02 4.47785264e-02
3.15945054e-02 2.89818053e-02 2.56077224e-02 2.31653929e-02
2.29037562e-02 2.09601611e-02 2.04041549e-02 1.98863970e-02
1.95361376e-02 1.87542029e-02 1.83088901e-02 1.80014833e-02
1.74602703e-02 1.69709530e-02 1.67357377e-02 1.63448269e-02
1.60331440e-02 1.59727699e-02 1.57128219e-02 1.52194510e-02
1.49060538e-02 1.46303692e-02 1.44740815e-02 1.41838751e-02
1.39922291e-02 1.36667844e-02 1.33348205e-02 1.30998692e-02
1.23829396e-02 1.15179891e-02 1.12459708e-02 1.10565830e-02
1.02889056e-02 9.71386900e-03 9.54109549e-03 9.06477741e-03
8.57930967e-03 8.49031967e-03 7.89031518e-03 7.61410315e-03
6.33842151e-03 6.10905281e-03 5.09039505e-03 4.80000820e-03
4.74912524e-03 4.44308189e-03 4.41109992e-03 4.04440878e-03
3.82977936e-03 3.66316563e-03 3.34563591e-03 2.58007217e-03
2.35489160e-03 2.05500573e-03 1.90188807e-03 1.36270244e-03
1.21163202e-03 2.41712682e-04 1.86978346e-07 2.78705198e-32]
```

图 3-12：各个主成分的可解释方差贡献率

使用 np.cumsum 函数计算累积方差贡献率，并将其绘制成曲线图，如图 3-13 所示。在这里，将保留特征数量设为 20%，即当累计方差占比达到 0.72 时截止。由这个图可以看出，红线以下的属性个数为 24，它们就是要保留的属性。

接下来，删除 ID 列等不必要的列，并选择需要进行 PCA 处理的数值属性。设定要保留的主成分数量为 24，即将数据降至 24 维。最后将主成分分析结果转换为 DataFrame 对象即可。

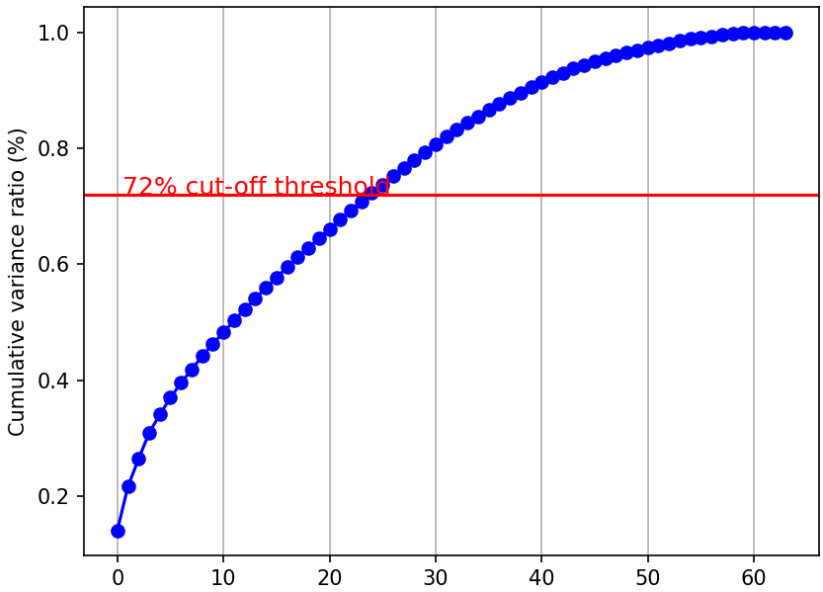

图 3-13：累计方差贡献率示意图

主成分分析中每个主成分解释数据的方差百分比如图 3-14 所示。这 24 个主成分的重要性依次递减，但总体上最大化地保留了原始数据集的特征。

```
解释方差比例： [0.14354561 0.07681723 0.04827391 0.04548833 0.03209332 0.02943655
 0.02594378 0.02336172 0.02278187 0.0209017  0.02062318 0.01980111
 0.01911861 0.01876664 0.01824181 0.01816387 0.01764025 0.0171586
 0.01645636 0.01627608 0.01619549 0.01586522 0.01553746 0.01507726]
```

图 3-14：每个主成分解释数据的方差百分比

3.2.6 利用 Kmeans 算法实现聚类（二分类）

通过 PCA，我将属性降维到了 24 维，下面将训练集中的 1460 条数据进行聚类。聚类后，认为每个聚类（簇）中的数据具有高相似性，并在下一步对每个聚类单独进行训练和线性回归，由此得到测试集的预测结果。

首先通过聚类算法的 Elbow Method 确定最优聚类数量。Elbow Method 是一种常用的聚类分析方法，其基本思想是通过求解不同的聚类数对应的簇内平方和（SSE）或其它衡量指标，找到一个“拐点”。在这个拐点之前，随着聚类数目的增加，模型的性能提升较为明显，但在这个点之后，再增加聚类数并不能显著地提高模型的性能表现，因此这个拐点通常被认为是最优聚类数目的估计值。该方法可以帮助用户选择合适的聚类数目，并可以避免聚类数量过多或者过少的问题。

在本实验中，我使用 Elbow Method 时引入了两种评价指标：每个簇的质点与簇内样本点的距离误差平方和（distortions）和样本距离最近的聚类中心的距离误差总和（inertias）。二者的计算公式分别是：

$$distortions = \sum_{\text{聚类中所有点}} \sum_{1 \le k \le 24} (x_{ik} - x_{0k})^2$$

$$inertias = \sum_{\text{聚类中所有点}} \sum_{1 \le k \le 24} |x_{ik} - x_{0k}|$$

其中，x为聚类中的点，x_{ik}表示第i个点在第k维上的坐标，x_0为聚类中心点。

实际上，前者表现出的效果更好，因为它对离聚类中心点越远的点的惩罚越大。具体实现时采用欧式距离计算即可，逐个计算聚类数量在 1~15 时的距离误差平方和。核心代码如下：

```
inertias = []
distortions = []
K = range(1, 15)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(X)
    kmeanModel.fit(X)
    distortions.append(sum(np.min(cdist(X, kmeanModel.cluster_centers_,
'euclidean'), axis=1)) / X.shape[0])
    inertias.append(kmeanModel.inertia_)
```

结果如图 3-15 所示。可见，k=2 为明显的拐点，因此本问题中最优聚类数量为 2。

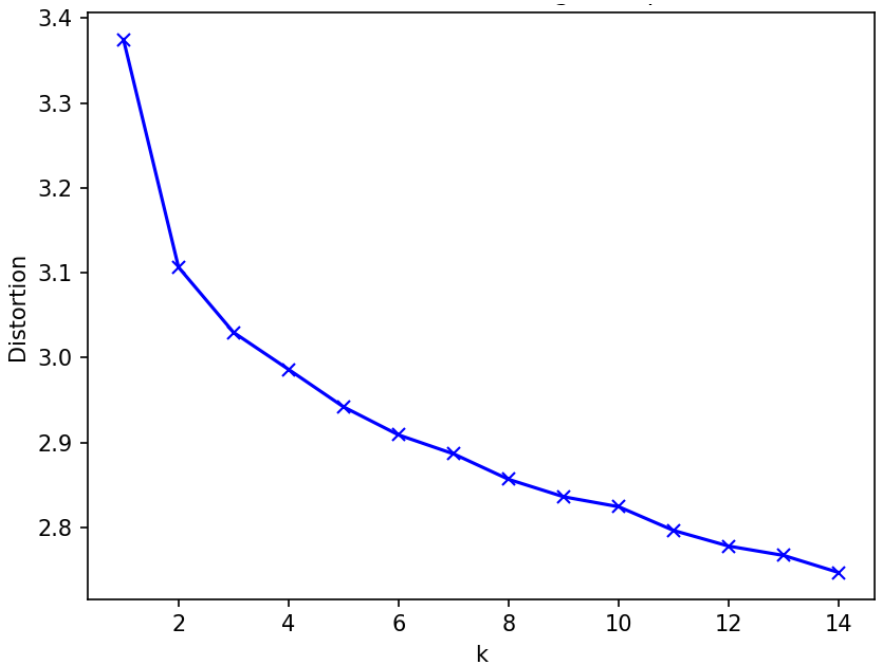

图 3-15：以 Distortion 为评价指标，利用 Elbow Method 确定最优聚类数量

据此即可对数据集应用 Kmeans 聚类算法进行二分类（n_clusters=2），将数据

集分为两个互不重叠的簇。然后用 predict 方法获取每个数据点所属的簇。最后，将全部数据分别保存到 cluster_0/cluster_1 两个 DataFrame 中，并输出各自的数量，如表 3-2 所示。

表 3-2：两个聚类的元素数量

聚类	元素数量
1	1528
2	1391

两个聚类中的元素数量近似相同，说明聚类具有均匀性。

3.2.7 训练、线性回归并得到预测结果

从 sklearn.linear_model 库中引入线性回归函数 LinearRegression，并对两个聚类分别进行训练和线性回归预测。

对于每个聚类，根据 Saleprice 是否为空来确定该行数据进入训练集还是测试集。将训练集和测试集分别存储在 train_df 和 test_df 中，并提取出 Id 列用于最终结果的合并。接着，分别从 train_df 和 test_df 中删除 Cluster 列（因为已经完成聚类，不再需要分类标签）、Id 列、Saleprice 列，并将剩余特征值作为 X_train 和 X_test，Saleprice 列作为 Y_train 和 Y_test。对 X_train 和 Y_train 进行线性回归拟合得到预测值，并将其与对应的 Id 存储起来。最后，将聚类结果和预测结果合并在一起，形成 Saleprice 的 DataFrame 并按照 Id 排序，将其保存为 csv 文件并输出为 saleprice.csv。

至此，已经得到了预测结果，见“结果与分析”章节。

综上所述，大作业的算法流程图如图 3-16 所示。

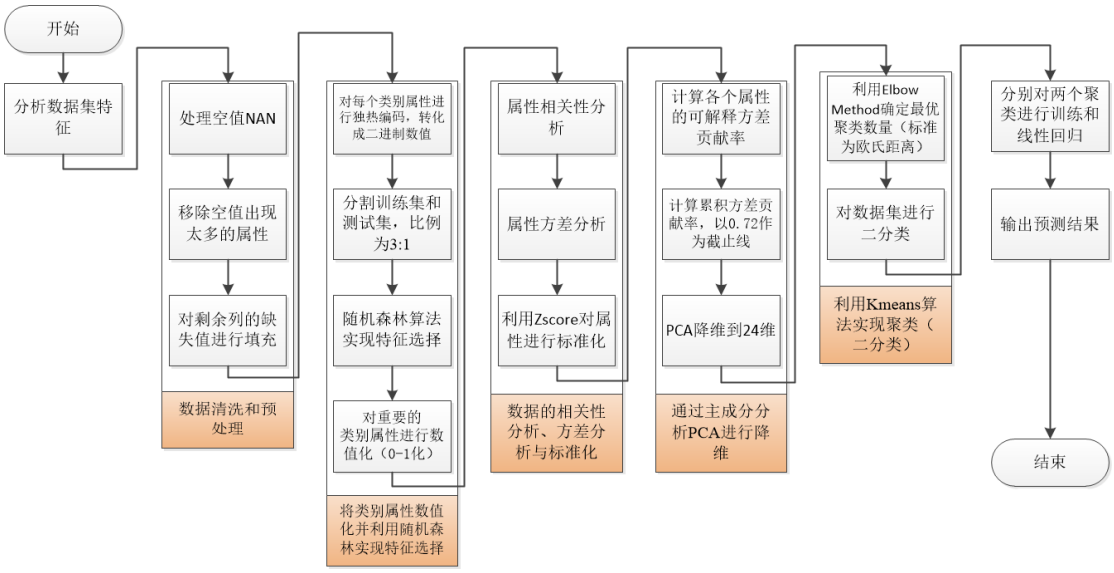

图 3-16：大作业算法流程图

3.3 实验环境与平台

实验平台是 Windows 11 操作系统下的 PyCharm Community 2023.1（64 bit）。

3.4 结果与分析

运行 ML_HousingRegression.py，程序生成预测结果 saleprice.csv，如图 3-17 所示。

	A	B
1	Id	SalePrice
2	1461	127910.1
3	1462	170991.8
4	1463	196611.2
5	1464	227349.1
6	1465	167998.2
7	1466	185648.2
8	1467	186996.8
9	1468	171772.2
10	1469	209853.1
11	1470	126184.4
12	1471	202001.5
13	1472	106433.8
14	1473	104864.1
15	1474	141135.6

图 3-17：房价预测结果（节选）

将结果上传到 Kaggle 平台评测，得分为 0.13597，预测效果好，如图 3-18 所示。

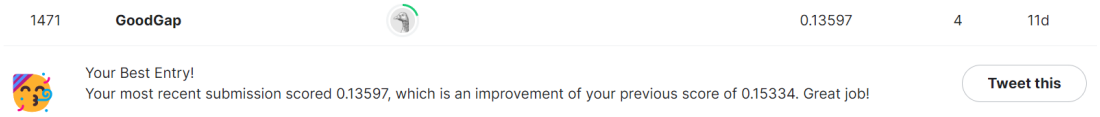

图 3-18：结果评测

3.5 个人体会

这是本学期机器学习课程的大作业，我主要是实现了对房价的预测。整个流程并非一帆风顺，而是遇到了一些问题，不过经过个人努力都得以解决。例如一开始没有将训练集和测试集进行分割，导致出现了过拟合，后来找到问题并将其以 3:1 的比例分割就能解决。还有，一开始在 PCA 降维时，不知道降到多少维才能较好地保留数据集原始特征。经过不断调参，最终发现截止线在 0.72 左右时效果较好，太高后降维效果比较差，太低了则会丢失很多重要信息。

这个过程中，根据最后的预测效果，我还进行了一些灵活的优化，例如一开始的预测效果不太好，且训练时间较长，后来我发现了有些属性对结果影响很大，

有些却影响不大。根据这一发现，启发我引入了随机森林和主成分分析 PCA 筛选出重要的属性并进行降维。同时我还用了机器学习领域中的多种方法，例如 Kmeans 聚类、线性回归等经典算法，进一步提高了我综合运用多种思路解决问题的能力，以及具体问题具体分析的思维。

总的来说，这学期我学到了很多关于机器学习的知识和方法。此前我从未真正上手实践过机器学习的算法，但我发现这是一门非常有趣和实用的学科，它可以帮助我更好地理解数据，并从数据中发现有用的规律和趋势。同时，我也深刻认识到了机器学习在实际应用中的问题和挑战，例如如何选择合适的算法、如何避免过拟合等等。总之，通过这学期的学习，我更加深入地了解了机器学习的本质和应用，收获颇丰。

附录一　实验六代码

（一）鸢尾花三分类问题 yuanweihua_experiment.py

```
import os
# os.environ['DEVICE_ID'] = '0'
import csv
import numpy as np
from pprint import pprint
import random

import mindspore as ms
from mindspore import nn
# from mindspore import context
from mindspore import dataset
from mindspore.train.callback import LossMonitor
from sklearn.datasets import load_iris

def create_dataset(data_path):
    # Todo 每个类的前五个样本信息
    x_data = load_iris().data
    y_data = load_iris().target.reshape(150, 1)
    data = np.hstack((x_data, y_data))

    # Todo 分别将 Iris-setosa，Iris-versicolor，Iris-virginica 对应为 0，1，2 三类
    label_dict = {0: 0, 1: 1, 2: 2}
    X = np.array([[float(x) for x in s[:-1]] for s in data[:150]],
np.float32)
    Y = np.array([label_dict[s[-1]] for s in data[:150]], np.int32)

    # Todo Using random choice and split dataset into train set and
validation set by 8:2.
    train_idx = np.random.randint(0, len(Y), int(len(Y) * 0.8))
    test_idx = np.array([i for i in range(0, len(Y)) if i not in train_idx])
    X_train, Y_train = X[train_idx], Y[train_idx]
    X_test, Y_test = X[test_idx], Y[test_idx]

    # Convert the training data to MindSpore Dataset.
    XY_train = list(zip(X_train, Y_train))
    ds_train = dataset.GeneratorDataset(XY_train, ['x', 'y'])
    ds_train = ds_train.shuffle(buffer_size=120).batch(32,
drop_remainder=True)

    # Convert the test data to MindSpore Dataset.
    XY_test = list(zip(X_test, Y_test))
    ds_test = dataset.GeneratorDataset(XY_test, ['x', 'y'])
```

```
    ds_test = ds_test.shuffle(buffer_size=120).batch(32, drop_remainder=True)
    return ds_train, ds_test

def softmax_regression(ds_train, ds_test):
    net = nn.Dense(4, 3)

    # Todo 使用交叉熵损失计算
    loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
    # Todo 使用动量优化器优化参数，其中学习率设置为 0.05，动量设置为 0.9
    opt = nn.optim.Momentum(net.trainable_params(), learning_rate=0.05,
momentum=0.9)

    model = ms.train.Model(net, loss, opt, metrics={'acc', 'loss'})
    model.train(25, ds_train,
callbacks=[LossMonitor(per_print_times=ds_train.get_dataset_size())],
dataset_sink_mode=False)
    metrics = model.eval(ds_test)
    print(metrics)

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_url', default='iris.data', help='Location of
data.')
    args, unknown = parser.parse_known_args()

    if args.data_url.startswith('s3'):
        # Todo:设置路径
        data_path = ""
        import moxing
        moxing.file.copy_parallel(src_url=os.path.join(args.data_url,
'iris.data'), dst_url=data_path)
    else:
        data_path = os.path.abspath(args.data_url)

    softmax_regression(*create_dataset(data_path))
```

（二）MNIST 手写数字识别 mnist_experiment.py

```
import os
import argparse
from mindspore import context
from mindspore.train.callback import Callback, LossMonitor

parser = argparse.ArgumentParser(description='MindSpore LeNet Example')
parser.add_argument('--device_target', type=str, default="CPU",
choices=['Ascend', 'GPU', 'CPU'])

args = parser.parse_known_args()[0]
# context.set_context(mode=context.GRAPH_MODE,
device_target=args.device_target)

import os
import requests

requests.packages.urllib3.disable_warnings()

def download_dataset(dataset_url, path):
    filename = dataset_url.split("/")[-1]
    save_path = os.path.join(path, filename)
    if os.path.exists(save_path):
        return
    if not os.path.exists(path):
        os.makedirs(path)
    res = requests.get(dataset_url, stream=True, verify=False)
    with open(save_path, "wb") as f:
        for chunk in res.iter_content(chunk_size=512):
            if chunk:
                f.write(chunk)
    print("The {} file is downloaded and saved in the path {} after
processing".format(os.path.basename(dataset_url), path))

train_path = "datasets/MNIST_Data/train"
test_path = "datasets/MNIST_Data/test"

download_dataset("https://mindspore-
website.obs.myhuaweicloud.com/notebook/datasets/mnist/train-labels-idx1-
ubyte", train_path)
download_dataset("https://mindspore-
website.obs.myhuaweicloud.com/notebook/datasets/mnist/train-images-idx3-
ubyte", train_path)
download_dataset("https://mindspore-
```

```
website.obs.myhuaweicloud.com/notebook/datasets/mnist/t10k-labels-idx1-
ubyte", test_path)
download_dataset("https://mindspore-
website.obs.myhuaweicloud.com/notebook/datasets/mnist/t10k-images-idx3-
ubyte", test_path)

import mindspore.dataset as ds
import mindspore.dataset.transforms.c_transforms as C
import mindspore.dataset.vision.c_transforms as CV
from mindspore.dataset.vision import Inter
from mindspore import dtype as mstype

def create_dataset(data_path, batch_size=32,
repeat_size=1,num_parallel_workers=1):
    # 定义数据集
    mnist_ds = ds.MnistDataset(data_path)
    # Todo 设置放缩的大小
    resize_height = 32
    resize_width = 32
    # Todo 归一化
    rescale = 1/255.
    shift = 0.0
    rescale_nml = 1 / 0.3081
    shift_nml = -1 * 0.1307 / 0.3081

    # 定义所需要操作的 map 映射
    resize_op = CV.Resize((resize_height, resize_width),
interpolation=Inter.LINEAR)
    rescale_nml_op = CV.Rescale(rescale_nml, shift_nml)
    rescale_op = CV.Rescale(rescale, shift)
    hwc2chw_op = CV.HWC2CHW()
    type_cast_op = C.TypeCast(mstype.int32)

    # 使用 map 映射函数，将数据操作应用到数据集
    mnist_ds = mnist_ds.map(operations=type_cast_op, input_columns="label",
num_parallel_workers=num_parallel_workers)
    mnist_ds = mnist_ds.map(operations=[resize_op, rescale_op,
rescale_nml_op, hwc2chw_op], input_columns="image",
num_parallel_workers=num_parallel_workers)

    # Todo 进行 shuffle、batch、repeat 操作
    buffer_size = 10000
    mnist_ds = mnist_ds.shuffle(buffer_size=buffer_size)
    mnist_ds = mnist_ds.batch(batch_size, drop_remainder=True)
```

```
    mnist_ds = mnist_ds.repeat(repeat_size)

    return mnist_ds

import mindspore.nn as nn
from mindspore.common.initializer import Normal


# Todo 根据 LeNet5 网络结构神经网络
class LeNet5(nn.Cell):
    """
    Lenet 网络结构
    """
    def __init__(self, num_class=10, num_channel=1):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(num_channel, 6, 5, pad_mode='valid')
        self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
        self.fc1 = nn.Dense(16 * 5 * 5, 120, weight_init=Normal(0.02))
        self.fc2 = nn.Dense(120, 84, weight_init=Normal(0.02))
        self.fc3 = nn.Dense(84, num_class, weight_init=Normal(0.02))
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()

    def construct(self, x):
        # 使用定义好的运算构建前向网络
        x = self.max_pool2d(self.relu(self.conv1(x)))
        x = self.max_pool2d(self.relu(self.conv2(x)))
        x = self.flatten(x)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x

class StepLossAccInfo(Callback):
    def __init__(self, model, eval_dataset, steps_loss, steps_eval):
        self.model = model
        self.eval_dataset = eval_dataset
        self.steps_loss = steps_loss
        self.steps_eval = steps_eval

    def step_end(self, run_context):
        cb_params = run_context.original_args()
        cur_epoch = cb_params.cur_epoch_num
```

```
        cur_step = (cur_epoch-1)*1875 + cb_params.cur_step_num
        self.steps_loss["loss_value"].append(str(cb_params.net_outputs))
        self.steps_loss["step"].append(str(cur_step))
        if cur_step % 125 == 0:
            acc = self.model.eval(self.eval_dataset, dataset_sink_mode=False)
            self.steps_eval["step"].append(cur_step)
            self.steps_eval["acc"].append(acc["Accuracy"])

# 实例化网络
net = LeNet5()

# MindSpore 支持的损失函数有 SoftmaxCrossEntropyWithLogits、L1Loss、MSELoss 等。
net_loss = nn.SoftmaxCrossEntropyWithLogits (sparse=True, reduction='mean')

# MindSpore 支持的优化器有 Adam、AdamWeightDecay、Momentum 等。这里使用 Momentum 优化
器为例。
net_opt = nn.Momentum (net.trainable_params(), learning_rate=0.01,
momentum=0.9)

from mindspore.train.callback import ModelCheckpoint, CheckpointConfig

config_ck = CheckpointConfig(save_checkpoint_steps=1875,
keep_checkpoint_max=10)

ckpoint = ModelCheckpoint(prefix="checkpoint_lenet", config=config_ck)

from mindspore.nn import Accuracy
from mindspore.train.callback import LossMonitor
from mindspore import Model

def train_net(model, epoch_size, data_path, repeat_size, ckpoint_cb,
sink_mode):
    """定义训练的方法"""
    # 加载训练数据集
    ds_train = create_dataset(os.path.join(data_path, "train"), 32,
repeat_size)
    eval_dataset = create_dataset(os.path.join(mnist_path, "test"), 32)
    steps_loss = {"step": [], "loss_value": []}
    steps_eval = {"step": [], "acc": []}
    step_loss_acc_info = StepLossAccInfo(model, eval_dataset, steps_loss,
steps_eval)
    model.train(epoch_size, ds_train, callbacks=[ckpoint_cb,
LossMonitor(125), step_loss_acc_info],
                dataset_sink_mode=sink_mode)
```

```
def test_net(model, data_path):
    """定义验证的方法"""
    param_dict = load_checkpoint("checkpoint_lenet-1_1875.ckpt")
    load_param_into_net(net, param_dict)
    ds_eval = create_dataset(os.path.join(mnist_path, "test"))
    acc = model.eval(ds_eval, dataset_sink_mode=False)
    print("Acc:{}".format(acc))

train_epoch = 1
mnist_path = "./datasets/MNIST_Data"
dataset_size = 1
model = Model(net, net_loss, net_opt, metrics={"Accuracy": Accuracy()})
train_net(model, train_epoch, mnist_path, dataset_size, ckpoint, False)
test_net(model, mnist_path)

from mindspore import load_checkpoint, load_param_into_net
# 加载已经保存的用于测试的模型
param_dict = load_checkpoint("checkpoint_lenet-1_1875.ckpt")
# 加载参数到网络中
load_param_into_net(net, param_dict)

import numpy as np
from mindspore import Tensor

# 定义测试数据集，batch_size 设置为 1，则取出一张图片
ds_test = create_dataset(os.path.join(mnist_path, "test"),
batch_size=1).create_dict_iterator()
data = next(ds_test)

# images 为测试图片，labels 为测试图片的实际分类
images = data["image"].asnumpy()
labels = data["label"].asnumpy()

# 使用函数 model.predict 预测 image 对应分类
output = model.predict(Tensor(data['image']))
predicted = np.argmax(output.asnumpy(), axis=1)

# 输出预测分类与实际分类
print(f'Predicted: "{predicted[0]}", Actual: "{labels[0]}"')
```

附录二　大作业代码

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statistics as st
import missingno as msno
import warnings
warnings.filterwarnings("ignore")

missing_values_set = ["", "--", "?", "na", "NAN", "nan", '']        # 将这些关
键字都视为空值（NAN）
test_data = pd.read_csv("./test.csv", na_values=missing_values_set)
train_data = pd.read_csv("./train.csv", na_values=missing_values_set)

# 绘制训练集数值属性相关性示意图
correlation_train = train_data.corr()
mask = np.zeros_like(correlation_train)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(correlation_train, mask=mask,
xticklabels=correlation_train.columns,
yticklabels=correlation_train.columns, linewidths=0.6, cmap='Blues')
plt.title("训练集中的数值属性相关性示意图")
plt.show()

frames = [train_data, test_data]
full_dataset = pd.concat(frames, sort=False, ignore_index=True)      # 将训练
集和测试集合并
full_dataset = pd.DataFrame(data=full_dataset)
SalePrice = full_dataset["SalePrice"]
full_dataset.drop(columns=['SalePrice'])
SalePrice.fillna("", inplace=True)              # 将空值变为""
ID = pd.DataFrame(data=full_dataset["Id"])

# 绘制每列数据的密集程度图（非空）
msno.matrix(full_dataset, labels=True)
plt.xticks(rotation='vertical', fontsize=10)
plt.title("各属性的空值占比（NAN）示意图")
plt.show()

# print(full_dataset.info())

# 数据预处理 preprocessing，搜索数据集中空值超过 600 个的列并删去
many_empty = [attribute for attribute in full_dataset if
```

```
full_dataset[attribute].isnull().sum() > 600] ##['Alley', 'FireplaceQu',
'PoolQC', 'Fence', 'MiscFeature']
full_dataset = full_dataset.drop(columns=many_empty)

# 获得数值和类别（非数值）属性
numerical_attr = [i for i in full_dataset.select_dtypes(include='number')]
categorical_attr = [i for i in full_dataset.select_dtypes(include='object')]

# 填充剩余列的缺失值。数值属性用这一列的平均值填充，类别属性用这一列出现次数最多的属性填充
for attributes in full_dataset:
    if attributes in numerical_attr:
        full_dataset[attributes] =
full_dataset[attributes].fillna(value=full_dataset[attributes].median())
    else:
        full_dataset[attributes] =
full_dataset[attributes].fillna(value=full_dataset[attributes].value_counts(
).idxmax())

# 下面会将类别属性数值化，并利用 RFC 和 SelectFromModel 筛选出重要的非数值属性
from sklearn.ensemble import RandomForestClassifier    # 随机森林分类器
from sklearn.feature_selection import SelectFromModel   # 特征选择
from sklearn.model_selection import train_test_split    # 避免 overfitting，只
关注训练集

X = pd.DataFrame(data=train_data[categorical_attr])
X.drop(columns=['SalePrice'], inplace=True)
# print(X)
Y = train_data['SalePrice']      # X 和 Y 是训练集的属性，其中 X 不含售价，Y 只含售价

for at in X:
    X = pd.concat([X, pd.get_dummies(X[at], prefix=at, drop_first=True)],
axis=1, sort=False)    # 利用 get_dummies 函数将类别属性转化为编码
    X.drop(columns=[at], inplace=True)

X_train_st, X_test_st, y_train_st, y_test_st = train_test_split(X, Y,
test_size=0.25)        # 对 X 和 Y 分别划分训练集和测试集，比例为 3:1

# 特征选择
# 下面首先对类别属性进行操作
rfc = RandomForestClassifier(n_estimators=150)         # 创建 150 棵树进行随机森林
处理
sel = SelectFromModel(estimator=rfc, threshold='mean')  # 利用 RFC 作为估计器，选
取重要度大于全部重要度平均值的属性
sel.fit(X, Y)
```

```
print("门限 threshold: ", sel.estimator_.feature_importances_.mean())

# print(sel.get_support())                                    # 打印属性是否高于 threshold 的矩阵（T/F）
selected_features = X_train_st.columns[(sel.get_support())]          # 获取属性名称及取值
print("高于门限值的类别属性取值个数：", len(selected_features), "\n 这些属性的值是：")
print(selected_features)

importances = sel.estimator_.feature_importances_             # 获取类别属性特征的重要度并从高到低排序
Features_importance = pd.DataFrame(data=X.columns.values,
columns=['Features'])
Features_importance['Importance'] = importances
Features_importance = Features_importance.sort_values(by=['Importance'],
axis=0, ascending=False)
print("这些类别属性取值的重要度：")
print(Features_importance)

print("全部类别属性取值的重要度：", importances)
indices = np.argsort(importances)[::-1]               # 返回需要对数组进行排序的索引

plt.figure(figsize=(10, 10))
plt.title("从高到低排列的类别属性取值的重要度")
plt.bar(range(X.shape[1]), importances[indices], color="r", align="center")
plt.xticks(range(X.shape[1]), Features_importance['Features'],
rotation='vertical')
plt.xlim([-1, 9.5])
plt.show()

# 将排名前十的类别属性数值化
cat_to_num = ["GarageFinish", "LotShape", "BsmtExposure", "BsmtFinType1",
"LotConfig", "HouseStyle", "MasVnrType"]
for categoricals in cat_to_num:
    full_dataset = pd.concat([full_dataset,
pd.get_dummies(full_dataset[categoricals], prefix=categoricals,
drop_first=True)], axis=1, sort=False)
    full_dataset.drop(columns=[categoricals], inplace=True)
    print("类别属性：", categoricals)       # 分别将上述类别属性数值化并展示结果
    print(full_dataset)

numerical_attr = [i for i in full_dataset.select_dtypes(include='number')]
```

```
# 绘制训练集的完整属性相关性示意图
correlation = full_dataset.corr()
mask = np.zeros_like(correlation)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(correlation, mask=mask, xticklabels=correlation.columns,
yticklabels=correlation.columns, linewidths=0.6, cmap='Blues')
plt.title('训练集的完整属性相关性示意图')
plt.show()

# 相关性分析
correlation['CORR'] = correlation.sum(axis=0)                # 计算每种属性与
saleprice 的相关系数，并创建一个名为 CORR 的列
feature_correlation = correlation['CORR'].sort_values(ascending=False)
print("属性相关性分析：\n", feature_correlation)

# 方差分析
vanalysis=[]
for attribute in numerical_attr:
    mean = st.mean(full_dataset[attribute])
    # print('\n head of {} is : {} '.format(attribute,
full_dataset[attribute].head()))
    # full_dataset[attribute].value_counts().plot(kind='hist', bins=20)   #绘
制每种属性的值与出现频率的直方图
    # plt.title(attribute)
    # plt.show()
    standard_variance = st.stdev(full_dataset[attribute])     # 均值方差标准化
    variance = st.variance(full_dataset[attribute])
    vanalysis.append([attribute, mean, variance, standard_variance])

vanalysis = pd.DataFrame(vanalysis, columns=['属性', '平均值', '方差', '标准化方
差']).sort_values(by=['标准化方差'], ascending=False)
print("方差分析：\n", vanalysis)

# 使用 Z-score 实现数据标准化
from scipy.stats import zscore

old_full_dataset = full_dataset
full_dataset[numerical_attr] = zscore(full_dataset[numerical_attr])       #
对目前的数据集计算 zscore

# PCA 对数据进行降维处理
from sklearn.decomposition import PCA
pca_search = PCA().fit(full_dataset[numerical_attr])       # fit pca
```

```
fig, ax = plt.subplots()
print("PCA 的各特征（可解释性）方差贡献率：\n",
pca_search.explained_variance_ratio_)
y = np.cumsum(pca_search.explained_variance_ratio_)
xi = np.arange(0, len(y), step=1)                              # 特征数量
plt.plot(xi, y, marker='o', color='b')
plt.xlabel('成分数量')
plt.ylabel('Cumulative variance ratio (%)')      # 累计方差贡献率，取 0.72 之前的属
性
plt.title('The number of components after dimension reduction')
plt.axhline(y=0.72, color='r', linestyle='-')
plt.text(0.5, 0.72, '72% cut-off threshold', color='red', fontsize=12)
ax.grid(axis='x')
plt.show()

data_to_pca = full_dataset.drop(columns=["Id"])        # 舍去 id 属性
numerical_attr_pca = [i for i in
data_to_pca.select_dtypes(include='number')]       # 获取 PCA 处理后的数值属性

pca = PCA(n_components=24)              # 降维后的维度为 24，即保留 24 种特征
principal_analysis = pca.fit_transform(data_to_pca[numerical_attr_pca])
# 拟合
distortions = []
principal_analysis = pd.DataFrame(data=principal_analysis, columns=['P1',
'P2', 'P3', 'P4', 'P5', 'P6', 'P7', 'P8', 'P9', 'P10', 'P11', 'P12', 'P13',
'P14', 'P15', 'P16', 'P17', 'P18', 'P19', 'P20', 'P21', 'P22', 'P23',
'P24'])    # 创建 dataframe
print("主成分分析（PCA）：\n ", principal_analysis)
print("解释方差比例：", pca.explained_variance_ratio_)

from scipy.spatial.distance import cdist
from sklearn.cluster import KMeans

#计算聚类数量
inertias = []                   # 样本距离最近的聚类中心的总和
distortions = []                # 每个簇的质点与簇内样本点的平方距离误差和，即畸变程度
K = range(1, 15)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(X)
    kmeanModel.fit(X)
    distortions.append(sum(np.min(cdist(X, kmeanModel.cluster_centers_,
'euclidean'), axis=1)) / X.shape[0])
    inertias.append(kmeanModel.inertia_)
```

```
# 利用 kmeans 算法的 elbow method 原理确定合适的聚类数量
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method')
plt.show()
#elbow with inertias:
# plt.plot(K, inertias, 'bx-')
# plt.xlabel('k')
# plt.ylabel('inertias')
# plt.title('The Elbow Method')
# plt.show()

pca_data = principal_analysis
clustering = KMeans(n_clusters=2, random_state=4)        # 二分类
clustering.fit(pca_data)

predict = clustering.predict(pca_data)
pca_data["Cluster"] = predict                # 在 pca_data 中创建三列：聚类（0 或
1）、Id、售价
pca_data["Id"] = ID.values
pca_data["SalePrice"] = SalePrice.values

cluster_0 = pd.DataFrame(data=pca_data[pca_data['Cluster'] == 0],
columns=['P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7', 'P8', 'P9', 'P10', 'P11',
'P12', 'P13', 'P14', 'P15', 'P16', 'P17', 'P18', 'P19', 'P20', 'P21', 'P22',
'P23', 'P24', 'Id', 'Cluster', 'SalePrice'])
cluster_1 = pd.DataFrame(data=pca_data[pca_data['Cluster'] == 1],
columns=['P1', 'P2', 'P3', 'P4', 'P5', 'P6', 'P7', 'P8', 'P9', 'P10', 'P11',
'P12', 'P13', 'P14', 'P15', 'P16', 'P17', 'P18', 'P19', 'P20', 'P21', 'P22',
'P23', 'P24', 'Id', 'Cluster', 'SalePrice'])

print("第一个聚类的数量：", len(cluster_0))
print("第二个聚类的数量：", len(cluster_1))

# 对两个聚类分别进行训练和线性回归
from sklearn.linear_model import LinearRegression
result = []
id = []
clusters = [cluster_0, cluster_1]

for i in range(0, 2):
    train = []
```

```
    test = []
    for row in clusters[i].values:
        x = row[26]        # SalePrice
        if x == "":              # 根据是否已有saleprice，确定每一行进入训练集还是测
试集
            test.append(row)
        else:
            train.append(row)

    train_df = pd.DataFrame(train, columns=['P1', 'P2', 'P3', 'P4', 'P5',
'P6', 'P7', 'P8', 'P9', 'P10', 'P11', 'P12', 'P13', 'P14', 'P15', 'P16',
'P17', 'P18', 'P19', 'P20', 'P21', 'P22', 'P23', 'P24', 'Id', 'Cluster',
'SalePrice'])
    test_df = pd.DataFrame(test, columns=['P1', 'P2', 'P3', 'P4', 'P5', 'P6',
'P7', 'P8', 'P9', 'P10', 'P11', 'P12', 'P13', 'P14', 'P15', 'P16', 'P17',
'P18', 'P19', 'P20', 'P21', 'P22', 'P23', 'P24', 'Id', 'Cluster',
'SalePrice'])
    Id = pd.DataFrame(data=test_df["Id"])

    train_df.drop(columns=["Cluster"], inplace=True)
    test_df.drop(columns=["Cluster"], inplace=True)

    X_train = train_df.drop(columns=["Id", "SalePrice"])
    Y_train = train_df["SalePrice"]
    X_test = test_df.drop(columns=["Id", "SalePrice"])
    Y_test = test_df["SalePrice"]

    regression = LinearRegression(n_jobs=-1)
    regression.fit(X_train, Y_train)
    prediction = regression.predict(X_test)
    id.append(Id)
    result.append(prediction)

result_for_cluster0 = pd.DataFrame(data=result[0],
columns=["cluster_predict"])      # 第一个聚类的预测结果
id0 = id[0]
result_for_cluster0["Id"] = id0.values

result_for_cluster1 = pd.DataFrame(data=result[1],
columns=["cluster_predict"])      # 第二个聚类的预测结果
id1 = id[1]
result_for_cluster1["Id"] = id1.values

# 得到结果
```

```
saleprice = pd.concat([result_for_cluster0, result_for_cluster1], axis=0,
sort=False)
saleprice.sort_values(by=['Id'], inplace=True)
saleprice.rename(columns={"cluster_predict": "SalePrice"}, inplace=True)

saleprice = pd.DataFrame(data=saleprice, columns=["Id", 'SalePrice'])
saleprice.to_csv(r"./saleprice.csv", index=False)
```